A PROJECT REPORT

ON

WEATHER PREDICTION USING MACHINE LEARNING

Submitted in partial fulfillment of the requirements for the award of the degree in

BACHELOR OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY

BY

A. GAYATHRI	**(20NN1A1267)**
J. DIVYA	**(20NN1A1287)**
D.KOMALI	**(20NN1A1293)**
CH. MADHUPRIYA	**(20NN1A1275)**

Under the esteemed guidance of Mrs.G.Lakshmi Durga

DEPARTMENT OF INFORMATION TECHNOLOGY

VIGNAN'S NIRULA INSTITUTE OF TECHNOLOGY AND SCIENCE FOR WOMEN
(Approved by AICTE, NEW DELHI and Affiliated to JNTUK)
PEDAPALAKALURU, GUNTUR-522005

(2020-2024)

VIGNAN'S NIRULA INSTITUTE OF TECHNOLOGY AND SCIENCE FOR WOMEN

(Approved by AICTE, NEW DELHI and Affiliated to JNTUK)

PEDAPALAKALURU, GUNTUR-522005

(2020-2024)

DEPARTMENT OF INFORMATION TECHNOLOGY

CERTIFICATE

This is to certify that the project work entitled "WEATHER PREDICTION USING MACHINE LEARNING" is a bonafide work submitted by A.Gayathri(20NN1A1267), J.Divya(20NN1A1287), D.Komali(20NN1A1293), CH.MadhuPriya(20NN1A1275) from the Department of Information Technology in the partial fulfillment of the requirements forward of the degree of Bachelor of Technology in Information Technology from Vignan's Nirula Institute of Technology & Science for Women, Guntur.

Internal Guide

Mrs.G.Lakshmi Durga Krishna

Head of the Department

Dr.K.V.S.S.Rama

External Examiner

DECLARATION

We hereby declare that the work described in this project work, entitled **“**WEATHER PREDICTION USING MACHINE LEARNING**”** which is submitted by us in partial fulfillment for the award of Bachelor of Technology in the Department of Information Technology to the Vignan’s Nirula Institute of Technology & Science for women, affiliated to Jawaharlal Nehru Technological University Kakinada, Andhra Pradesh, is the result of work done by us under the esteemed guidance of Mrs.G.Lakshmi Durga.

The work is original and has not been submitted for any Degree/Diploma of this or any other university.

A. GAYATHRI	**(20NN1A1267)**
J. DIVYA	**(20NN1A1287)**
D.KOMALI	**(20NN1A1293)**
CH. MADHUPRIYA	**(20NN1A1275)**

ACKNOWLEDGEMENT

We are profoundly grateful to express our deep sense of gratitude and respect towards our honorable chairman**, LAVU RATHAIAH** sir, Chairman of Vignan group for his precious support in the college.

We are very thankful to **DR.P. RADHIKA**, Principal VNITSW, Guntur, for her support during and till the completion of the project.

We would like to thank **DR.K.V.S.S. RAMA KRISHNA**, Associate Professor and Head of the Department of Information Technology, for his extended and continuous support, valuable guidance, and timely advice in the completion of this project thesis.

We wish to express our profound sense of sincere gratitude to our Project Guide, **MRS.G.LAKSHMI DURGA**, Assistant Professor of Information Technology, without whose help, guidance, and motivation this project thesis could not have been completed the project successfully.

We also thank all the faculty of the Department of Information Technology for their help and guidance on numerous occasions, which has given us the cogency to build up adamant aspirations over the completion of our project thesis finally, we thank one and all who directly or indirectly helped us to complete our project thesis successfully.

A. GAYATHRI	**(20NN1A1267)**
J. DIVYA	**(20NN1A1287)**
D.KOMALI	**(20NN1A1293)**
CH. MADHUPRIYA	**(20NN1A1275)**

ABSTRACT

This study investigates the efficacy of ensemble techniques in enhancing weather prediction accuracy. Ensemble methods combine the forecasts of multiple models to generate more robust and reliable predictions. By leveraging the diversity of individual models, ensemble techniques mitigate errors and uncertainties inherent in single-model forecasts. Through experimental evaluations and case studies, we demonstrate the effectiveness of ensemble methods in improving forecast accuracy and providing valuable uncertainty estimates. Overall, this research highlights the potential of ensemble techniques as a powerful tool for advancing weather prediction capabilities and supporting informed decision-making in various sectors.

Weather prediction is a critical aspect of meteorology with far-reaching implications for various sectors, including agriculture, transportation, and disaster management. Traditional weather forecasting models often face challenges in accurately capturing the complex and dynamic nature of atmospheric processes. Ensemble techniques have emerged as a promising approach to improving the reliability and accuracy of weather predictions by leveraging the strengths of multiple forecasting models.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER-1
INTRODUCTION

CHAPTER-1

INTRODUCTION

Introduction

Weather prediction is a critical aspect of meteorology with far-reaching implications for various sectors, including agriculture, transportation, and disaster management. Traditional weather forecasting models often face challenges in accurately capturing the complex and dynamic nature of atmospheric processes. Ensemble techniques have emerged as a promising approach to improve the reliability and accuracy of weather predictions by leveraging the strengths of multiple forecasting models.

In recent years, the demand for accurate and reliable weather predictions has grown significantly, driven by the increasing impact of weather-related events on various aspects of society, including agriculture, transportation, energy production, and public safety. However, weather forecasting remains a challenging task due to the inherent complexity of atmospheric processes, the limited availability of observational data, and the presence of uncertainties in climate models.

To address these challenges, meteorologists and researchers have turned to ensemble techniques, which have gained popularity for their ability to improve the quality and reliability of weather forecasts. Ensemble methods involve combining predictions from multiple individual models to produce a consensus forecast that often outperforms any single model. These techniques leverage the diversity of models, capturing different aspects of the underlying physical processes and statistical relationships in the atmosphere, thereby enhancing the overall predictive skill.

Ensemble techniques encompass a variety of approaches, including model averaging, bagging, boosting, and stacking, each with its unique strengths and applications in weather prediction. Model averaging methods aggregate predictions by calculating the mean or median of individual model outputs, effectively reducing the impact of outliers and errors. Bagging techniques, such as random forests, build multiple models on bootstrap samples of the training data and combine their predictions through averaging or voting, thereby reducing variance and improving robustness. Boosting algorithms, such as gradient boosting machines, iteratively train weak learners to focus on the misclassified instances, leading to improved predictive performance. Stacking, a more advanced ensemble method, combines the predictions of multiple base models using a meta-learner, which learns to weigh the contributions of individual models based on their performance.

In recent years, the integration of machine learning techniques, such as deep learning and artificial neural networks, into ensemble frameworks has further advanced the capabilities of weather prediction models. These data-driven approaches can capture complex nonlinear relationships in the atmosphere and extract meaningful features from large-scale meteorological datasets, leading to enhanced forecasting accuracy.

This introduction provides an overview of the role of ensemble techniques in weather prediction, highlighting their importance in addressing the challenges of uncertainty and variability inherent in atmospheric processes. In the following sections, we will delve deeper into the principles, methodologies, and applications of ensemble methods in weather forecasting, examining case studies, research findings, and future directions in the field. By harnessing the collective intelligence of diverse models and algorithms, ensemble techniques offer a promising avenue for improving the accuracy and reliability of weather predictions, ultimately benefiting society by enabling better preparedness and decision-making in the face of changing weather conditions.

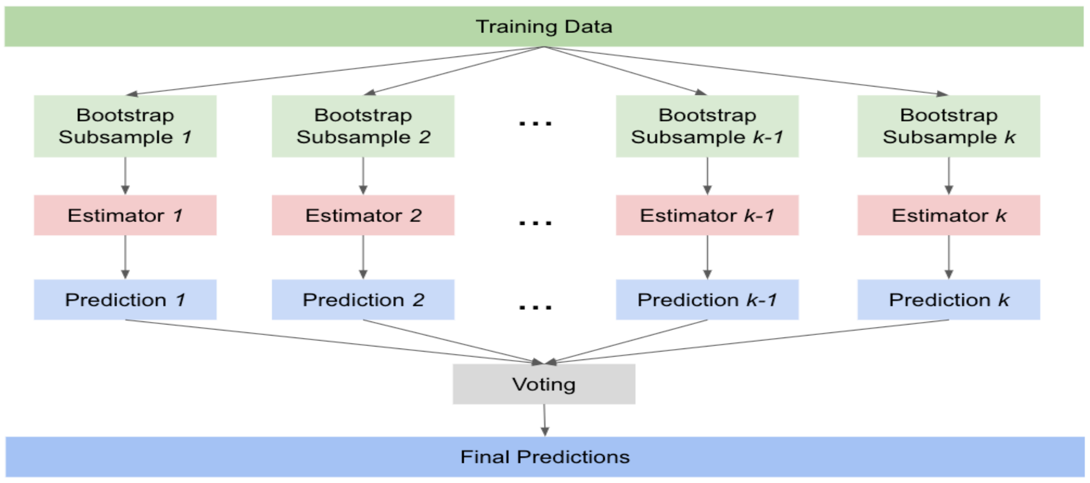


Fig.1: Training data

Algorithms Used

Random forest regression:

Random Forest Regression is a supervised learning algorithm that uses an ensemble learning method for regression. The ensemble learning method is a technique that combines predictions from multiple machine learning algorithms to make a more accurate prediction than a single model.

The Working process can be explained in the below steps and diagram:

- **Step-1:** Select random K data points from the training set.
- **Step-2:** Build the decision trees associated with the selected data points (Subsets).
- **Step-3:** Choose the number N for decision trees that you want to build.
- **Step-4:** Repeat Step 1 & 2.
- **Step-5:** For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.

Decision tree regression:

It is a tree-structured classifier with three types of nodes. The **Root Node** is the initial node which represents the entire sample and may get split further into further nodes. The **Interior Nodes** represent the features of a data set and the branches represent the decision rules. Finally, the **Leaf Nodes** represent the outcome. This algorithm is very useful for solving decision-related problems.

The complete process can be better understood using the below algorithm:

- **Step-1:** Begin the tree with the root node, says S, which contains the complete dataset.
- **Step 2:** Find the best attribute in the dataset using **Attribute Selection Measure (ASM).**
- **Step 3:** Divide the S into subsets that contain possible values for the best attributes.
- **Step 4:** Generate the decision tree node, which contains the best attribute.
- **Step-5:** Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and call the final node as a leaf node.

KNN algorithm:

The KNN algorithm assumes that similar things exist close. In other words, similar things are near to each other.

The K-NN working can be explained on the basis of the below algorithm:

- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean distance of **K number of neighbors**
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.
- **Step-6:** Our model is ready.

Support vector regression:

The Support Vector Regression (SVR) uses the same principles as the SVM for classification, with only a few minor differences.
SVMs can handle highly non-linear data using an amazing technique called kernel trick. It implicitly maps the input vectors to higher dimensional (adds more dimensions) feature spaces by the transformation which rearranges the dataset in such a way that it is linearly solvable.
Steps

- Load the dataset from source
- Separate input features and target variables.
- Buil and train the SVM classifiers using RBF kernel.
- Plot the scatter plot of the input features.
- Plot the decision boundary.

Gradient Boosting algorithm:

Gradient Boosting is a popular boosting algorithm. In gradient boosting, each predictor corrects its predecessor's error. Each predictor is trained using the residual errors of predecessor as labels. There is a technique called the Gradient Boosted Trees.

There are a few important steps in boosting the algorithm as follows:

- Consider a dataset having different data points and initialize it.
- Now, give equal weight to each of the data points.
- Assume this weight as an input for the model.
- Identify the data points that are incorrectly classified.
- Increase the weight for data points in step 4.
- If you get appropriate output then terminate this process else follow steps 2 and 3 again.

Voting Regressor Algorithm:

The voting algorithm has two variants: Voting Classifier and Voting Regressor. The voting classifier is explicitly used for only classification tasks, while the voting regressor is used for regression tasks, but both work in similar ways with few logical changes. This ensemble method combines the predictions from multiple individual regression models (traditional or other ensemble methods) to make a final prediction by leveraging the wisdom of the crowd by simple averaging or weighted averaging of the predictions of its constituent models, which leads to more accurate and robust predictions compared to individual models.

- Importing required libraries- We will import all required.
- Dataset loading and splitting- Now we will load the Dataset and split it into training and testing sets.
- Creating individual regression models- Before implementing the Voting regressor we need to define individual models that will work as estimators in the Voting algorithm.
- Voting Regressor model training- we will train the Voting regressor model.
- Model evaluation- After training, we will evaluate our model's performance.

Stacking Regressor algorithm:

Stacking is a way of ensembling classification or regression models it consists of two-layer estimators. The first layer consists of all the baseline models that are used to predict the outputs on the test datasets. The second layer consists of a Meta-Classifier or Regressor which takes all the predictions of baseline models as an input and generates new predictions.

- Importing required libraries- We will import all required.
- Dataset loading and splitting- Now we will load the Dataset and split it into training and testing sets.
- Building first Layer estimators- Train and evaluate with our first layer estimators to observe the difference in the performance of the stacked model and general model.
- Training each classifier of the layer of estimators.
- Predictions of each model are given as input features for the meta-classifier.
- Build the meta classifier and train that classifier.
- Evaluating the Meta-Classifier.

CHAPTER – 2
SYSTEM ANALYSIS AND DESCRIPTION

CHAPTER-2

SYSTEM ANALYSIS AND DESCRIPTION

2.1 EXISTING SYSTEM:

DISADVANTAGES OF THE EXISTING SYSTEM:

The accuracy and reliability of traditional weather prediction methods are often hindered by factors such as

- Complex weather patterns
- Lack of real-time data
- And limited understanding of the interactions between various atmospheric variables.

These challenges limit our ability to accurately forecast weather conditions, which can have significant implications for industries such as agriculture, transportation, and emergency management.

Traditional weather prediction models often rely on physical principles governing atmospheric dynamics and thermodynamics. These models simulate the behavior of the atmosphere using complex mathematical equations based on physical laws. They incorporate various factors such as temperature, pressure, humidity, wind speed, and direction to forecast weather conditions.

1. Approach:

Traditional Models: Based on physical principles and mathematical equations.

Machine Learning Models: Utilize historical data to learn patterns and relationships between weather variables and predictions.

2. Complexity:

Traditional Models: Highly complex, involving intricate mathematical equations and computational algorithms.

Machine Learning Models: Relatively simpler, focusing on learning patterns from data rather than explicitly modeling physical processes.

3. Data Requirements:

Traditional Models: Require high-quality observational data, such as temperature, pressure, humidity, wind observations, etc.

Machine Learning Models: Also require observational data but can handle a broader range of variables, including non-conventional data sources like satellite imagery, social media feeds, etc.

4. Training and Calibration:

Traditional Models: Require extensive calibration and tuning based on historical data and known physical relationships.

Machine Learning Models: Also require training on historical data but automate feature selection and model calibration.

5. Interpretability:

Traditional Models: Often more interpretable as they explicitly represent physical processes.

Machine Learning Models: Less interpretable, especially for complex models like neural networks, but techniques exist to interpret model predictions to some extent.

6. Performance:

Traditional Models: Can provide accurate forecasts, especially for short-term predictions and specific weather phenomena.

Machine Learning Models: Can also provide accurate forecasts and might excel in capturing complex nonlinear relationships, but their performance heavily depends on data quality, feature engineering, and model selection.

7. Computational Requirements:

Traditional Models: Typically require substantial computational resources, especially for high-resolution simulations.

Machine Learning Models: Can be computationally efficient, especially during prediction time, but training complex models with large datasets can be computationally intensive.

In summary, traditional weather prediction models and machine learning-based approaches each have their strengths and weaknesses. Traditional models offer physical interpretability and accuracy, especially for specific weather phenomena, while machine learning models can capture complex relationships in data and automate feature selection but may lack the physical interpretability of traditional models. The choice between the two depends on factors such as the specific forecasting task, data availability, computational resources, and the importance of interpretability. Often, a combination of both approaches can lead to more robust and accurate predictions.

2.2 PROPOSED SYSTEM

This code performs machine learning regression to predict temperature based on weather data. The data goes through several cleaning and pre-processing steps, including handling missing values, removing redundant features, addressing correlated features, creating new features from dates, and encoding categorical variables.

The code then trains and evaluates multiple regression models: Gradient Boosting Regressor, Random Forest Regressor, K-Nearest Neighbors Regressor, Decision Tree Regressor, and Support Vector Regressor. It also includes a Voting Regressor that combines predictions from all these models.

The final part evaluates the performance of each model using the R-squared score and Root Mean Squared Error (RMSE) on a hold-out test set. However, the code snippet doesn't explicitly show the selection of the best-performing model.

Overall, this code demonstrates a process for building and evaluating regression models to predict temperature based on weather data.

1. Importing Libraries:

- Necessary libraries for data manipulation (pandas - pd), machine learning (scikit-learn) and plotting (matplotlib) are imported.

2. Loading and Cleaning Data:

- The code reads weather data from a CSV file named "weatherHistory.csv" using pandas.
- It checks for missing values (null values) and fills them with appropriate strategies:
    - "Precip Type" is filled with the most frequent value (mode).
    - "Formatted Date" is converted to a datetime format.
- Features with low importance or redundancy are removed:
    - "Loud Cover" has only one value and is dropped.
    - "Daily Summary" is likely redundant for temperature prediction and dropped.
    - "Apparent Temperature" is highly correlated with "Temperature" and dropped.

3. Feature Engineering:

- New features like year, month, and day are extracted from the "Formatted Date" column.
- Textual features ("Summary" and "Precip Type") are converted to numerical values using LabelEncoder. This helps machine learning algorithms understand these features.
- The target variable ("Temperature (C)") is separated from the features stored in "X".
- Feature scaling is applied using StandardScaler. This ensures all features contribute equally to the model's learning process.

4. Splitting Data:

- The data is split into training and testing sets using train_test_split. The training set (70%) is used to train the models, and the testing set (30%) is used to evaluate their performance on unseen data.

5. Model Building and Evaluation:

- Multiple machine learning models are created:
    - Gradient Boosting Regressor (gbr)
    - Random Forest Regressor (rfr)
    - K Neighbors Regressor (knr)
    - Decision Tree Regressor (dtr)
    - Support Vector Regressor (svr)
- A Voting Regressor (vr) is created that combines the predictions of all the individual models for potentially better performance.
- The code checks for missing values again and replaces them with 0 (This might not be the best approach in all cases).
- All models are trained on the training data.
- The performance of each model (gbr, rfr, knr, dtr, svr, and vr) is evaluated on the testing data using R-squared (r2_score) and Root Mean Squared Error (RMSE).

6. Visualization of Results:

- The R-squared values for all models are plotted using matplotlib to visually compare their performance.

Overall, this code trains and compares different machine learning models to predict temperature based on weather data. It performs data cleaning, feature engineering, model building, evaluation, and visualization.

2.3 SOFTWARE DEVELOPMENT LIFE CYCLE:

There are various software development approaches defined and designed which are used/employed during the development process of software, these approaches are also referred as "Software Development Process Models". Each process model follows a particular life cycle in order to ensure success in process of software development.

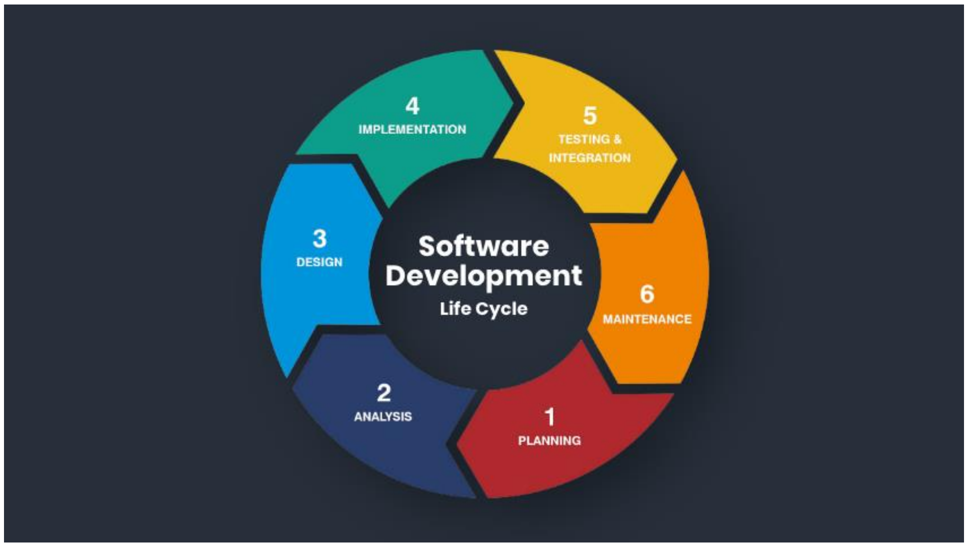


Fig.2: Software Development Life Cycle

REQUIREMENTS

Business requirements are gathered in this phase. This phase is the main focus of the project, how the weather is changing. Here we must predict the weather, so for this we need to determine the requirements. How is the weather changing day by day? How will they use the system? What data should be input into the system? What data should be output by the system? These are general questions that get answered during a requirement gathering phase. This produces a nice big list of functionalities that the system should provide, which describes functions the system should perform, business logic that processes data, what data is stored and used by the system,
and how the user interface should work. The overall result is the system as a whole and how it performs, not how it is actually going to do it.

DESIGN

Code is produced from the deliverables of the design phase during implementation, and this is the longest phase of the software development life cycle. For a developer, this is the main focus of the life cycle because this is where the code is produced. Implementation my overlap with both the design and testing phases. Many tools exists (CASE tools) to actually automate the production of code using information gathered and produced during the design phase.

TESTING

During testing, the implementation is tested against the requirements to make sure that the product is actually solving the needs addressed and gathered during the requirements phase. Unit tests and system/acceptance tests are done during this phase. Unit tests act on a specific component of the system, while system tests act on the system as a whole.

So in a nutshell, that is a very basic overview of the general software development life cycle model. Now let's delve into some of the traditional and widely used variations.

2.4 STUDY OF THE SYSTEM

In the flexibility of uses the interface has been developed a graphics concepts in mind, associated through a browser interface. The GUI's at the top level has been categorized as follows:

- Administrative User Interface Design
- The Operational and Generic User Interface Design

The administrative user interface concentrates on the consistent information that is practically, part of the organizational activities and which needs proper authentication for the data collection. The Interface helps the administration with all the transactional states like data insertion, data deletion, and data updating along with executive data search capabilities.

The operational and generic user interface helps the users upon the system in transactions through the existing data and required services. The operational user interface also helps the ordinary users in managing their own information helps the ordinary users in managing their own information in a customized manner as per the assisted flexibilities.

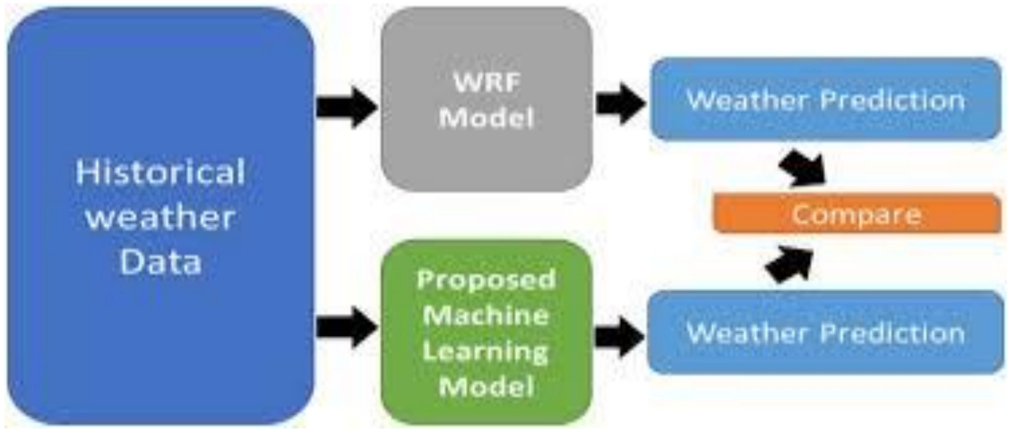


Fig.3: Study of the System

2.5 SYSTEM ANALYSIS

The System Development Life Cycle (SDLC), or Software Development Life Cycle in systems engineering, information systems and software engineering, is the process of creating or altering systems, and the models and methodologies that people use to develop these systems.

In software engineering the SDLC concept underpins many kinds of software development methodologies. These methodologies from the framework for planning and controlling the creation of an information system the software development process.

SOFTWARE MODEL OR ARCHITECTURE ANALYSIS:

Structured project management techniques (such as an SDLC) enhance management's control over projects by dividing complex tasks into manageable sections. A software life cycle model is either a descriptive or prescriptive characterization of how software is or should be developed. But none of the SDLC models discuss the key issues like Change management, Incident management and Release management processes within the SDLC process, but, it is addressed in the overall project management. In the proposed hypothetical model, the concept of user-developer interaction in the conventional SDLC model has been converted into a three-dimensional model which comprises of the user, owner and the developer. In the proposed hypothetical model, the concept of user-developer interaction in the conventional SDLC model has been converted into a three-dimensional model which comprises of the user, owner and the developer. The one size fits all‖ approach to applying SDLC methodologies is no longer appropriate. We have made an attempt to address the above-mentioned defects by using a new hypothetical model for SDLC described elsewhere. The drawback of addressing these management processes under the overall project management is missing of key technical issues pertaining to software development process that is, these issues are talked in the project management at the surface level but not at the ground level.

CHAPTER-3

REQUIREMENT SPECIFICATION

CHAPTER-3
REQUIREMENT SPECIFICATION

REQUIREMENT ANALYSIS

Analysis is defined as detailed examination of the elements or structure of something.

The process to gather the software requirements from clients, analyze and document them is known as requirements engineering or requirements analysis. The goal of requirement engineering is to develop and maintain sophisticated and descriptive 'System/Software Requirements Specification' documents. It is a four step process generally, which includes –

- Feasibility Study
- Requirements Gathering
- Software Requirements Specification
- Software Requirements Validation

The basic requirements of our project are:

- Research Papers
- Dataset

Operating System: The code can be run on any operating system supported by Python, such as Windows, macOS, or Linux.

Software Dependencies: Python 3.x: The code is written in Python programming language.

- NumPy: For numerical computations.

- Pandas: For data manipulation and analysis.

- Scikit-learn (sklearn): For machine learning algorithms and evaluation metrics.

- Matplotlib: For data visualization.

Performance: The system should be capable of handling large datasets efficiently.

Accuracy: The models should provide accurate temperature predictions.

Scalability: The system should be scalable to accommodate additional regression algorithms and features.

Usability: The system should have a user-friendly interface for inputting data and viewing predictions.

3.1FUNCTIONAL REQUIREMENTS ANALYSIS

Functional requirements explain what has to be done by identifying the necessary task, action or activity that must be accomplished. Functional requirements analysis will be used as the top-level functions for functional analysis.

USER REQUIREMENTS ANALYSIS

User Requirements Analysis is the process of determining user expectations for a new or modified product. These features must be quantifiable, relevant and detailed. The main user requirements of our project are as follows:

- Dataset
- Operating system
- Intel i3 preprocessor
- RAM 4 GB
- Hard disk 500GB
- python language and Google colab

3.2 NON FUNCTIONAL REQUIREMENTS ANALTSIS

Non-functional requirements describe the general characteristics of a system. They are also known as quality attributes. Some typical non-functional requirements are Performance, Response Time, Throughput, Utilization, and Scalability.

Performance:

The performance of a device is essentially estimated in terms of efficiency, effectiveness and speed.

- Short response time for a given piece of work.
- High throughput (rate of processing work)
- Short data transmission time.

Response Time: Response time is the time a system or functional unit takes to react to a given input.

FEASIBILITY STUDY:

Feasibility Study is a high level capsule version of the entire process intended to answer a number of questions like: What is the problem? Is there any feasible solution to the given problem? Is the problem even worth solving? Feasibility study is conducted once the problem is clearly understood. Feasibility study is necessary to determine that the proposed system is Feasible by considering the technical, Operational, and Economical factors. By having a detailed feasibility study the management will have a clear-cut view of the proposed system. A well designed feasibility study should provide a historical background of the business or project, the operations and management, marketing research and policies, financial data, legal requirements and tax obligations. The following feasibilities are considered for the project in order to ensure that the project is variable and it does not have any major obstructions. Feasibility study encompasses the following things:

- Technical Feasibility
- Operational Feasibility
- Behavioral feasibility

In this phase, we study the feasibility of all proposed systems, And pick the best feasible solution for the problem. The feasibility is studied based on three main factors as follows:

Technical feasibility:

In this step, we verify whether the proposed systems are technically feasible or not. i.e., all the technologies required to develop the system are available readily or not. Technical Feasibility determines whether the organization has the technology and skills necessary to carry out the project and how this should be obtained. The system can be feasible because of the following grounds.

- All necessary technology exists to develop the system.
- This system is flexible, and it can be expanded further.
- This system can give a guarantee of accuracy, ease of use, and reliability.
- Our project is technically feasible because all the technology needed for our project is readily available.

Operational feasibility:

In this step, we verify different operational factors of the proposed systems like manpower, time etc., whichever solution uses less operational resources, is the best operationally feasible solution. The solution should also be operationally possible to implement. Operational

Feasibility determines if the proposed system satisfied user objectives could be fitted into the current system operation.

- ➢ The methods of processing and presentation are completely accepted by the clients since they can meet all user requirements.
- ➢ The clients have been involved in the planning and development of the system.
- ➢ The proposed system will not cause any problem under any circumstances.
- ➢ Our project is operationally feasible because the time requirements and personnel requirements are satisfied.

We are a team of four members, and we worked on this project for three working months.

3.3 SOFTWARE AND HARDWARE REQUIREMENTS:

Software Requirements:

- Operating System: Windows 10.
- Coding Language: Python.
- Coding Environment: Google colab.

Hardware Requirements:

- Preprocessor: Intel Core i5.
- Hard Disk: 500 GB.
- RAM: 8GB.

SRS Specification:

Software Requirements specification (SRS) – a requirements specification for a software system- is a complete description of behavior of a system to be developed. It includes a set of cases that describe all the interactions users will have with the software. In addition to use cases, the SRS also contains non-functional requirements. Non-functional requirements are requirements which impose constraints on the design or implementation (such as performance engineering requirements, quality standards, or design constraints).

System Requirements Specification It is a collection of information that embodies the requirements of a system. A business analyst, sometimes titled system analyst, is responsible for analyzing the business needs of their clients and stakeholders to help identify business problems and propose solutions. Projects are subject to three sorts of required elements. Business requirements describe in business terms what must be delivered or accomplished to provide value.

- Product requirements describe properties of a system or product (which could be one of several ways to accomplish a set of business requirements.)
- Process requirements describe activities performed by the developing organization. For instance, process requirements could specify methodologies that must be followed, and constraints that the organization must obey.

Product and process requirements are closely linked. Process requirements often specify the activities that will be performed to satisfy a product requirement. For example, a maximum development cost requirement (a process requirement) may be imposed to help achieve a maximum sales price requirement (a product requirement) a requirement that the product be maintainable (a product requirement) often is addressed by imposing requirements to follow particular development styles. A system engineering requirement can be a description of what a system must do, referred to as Functional Requirement. This type of requirement specifies something that the delivered system must be able to do. Another type of requirement specifies something about the system itself, and how well it performs its functions. Such requirements are often called Nonfunctional requirements, or 'Performance requirements' or 'Quality ofservice requirements'. Examples of such requirements include usability, availability, reliability, supportability, testabilityand maintainability.

A collection of requirements define the characteristics or features of the desired system. A 'good' list of requirements asfar as possible avoids saying how the system should implement the requirements, leaving such decisions to the system designer. Specifying how the system should be implemented is called "implementation bias" or "solution engineering". However, implementation constraints on the solution may validly be expressed by the future owner, for example for required interfaces to external systems; for interoperabilitywith other systems; and for commonalitywith other owned products.

Functional requirements:

The Functional Requirements Specification gives the operations and activities that a system must be able to perform. Functional requirements should include functions performed by specific screens, outlines of work- flows performed by the system, and other business or compliance requirements the system must meet. It also depends upon the type of software, expected users and the type of system where the software is used.

A video was given as input, which is further divided into frames. The frame was given as input to detection code which detects the objects based on the training set. Count from each lane can be calculated and allocate signaling time dynamically. The output was in the form of signaling time.

Non functional requirements:

In systems engineering, a non-functional requirement is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. They are contrasted with functional requirements that define specific behavior or functions. The nonfunctional requirements can be considered as quality attributes of a system.
Performance: The time required to divide the video into frames.

Reliability: The system should be 90% reliable.

Since it may need some maintenance or preparation for some particular day, the system does not need to be reliable every time. So, 80% reliability is enough.
Availability: It is available in all the metropolitan cities.

Maintainability: The system should be optimized for supportability, or ease of maintenance as far as possible.

CHAPTER – 4
LANGUAGE OF IMPLEMENTATION

CHAPTER-4
LANGUAGES OF IMPLEMENTATION

4.1: PYTHON

What is Python?

Chances you are asking yourself this. You may have found this book because you want to learn to program but don't know anything about programming languages. Or you may have heard of programming languages like C, C++, C#, or Java and want to know what Python is and how it compares to "big name" languages. Hopefully I can explain it for you.

PYTHON CONCEPTS

If you're not interested in the how's and whys of Python, feel free to skip to the next chapter. In this chapter I will try to explain to the reader why I think Python is one of the best languages available and why it's a great one to start programming with.

- Open-source general-purpose language.
- Object Oriented, Procedural, Functional
- Easy to interface with C/ObjC/Java/Fortran
- Easy-ish to interface with C++ (via SWIG)

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently whereas other languages use punctuation, and it has fewer syntactical constructions than other languages.

- **Python is Interpreted** − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is like PERL and PHP.
- **Python is Interactive** − You can sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented** − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language** − Python is a great language for beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

HISTORY OF PYTHON

- Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.
- Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages.
- Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).
- Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

PYTHON FEATURES

- **Easy-to-learn** − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read** − Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain** − Python's source code is fairly easy-to-maintain.
- **A broad standard library** − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode** − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable** − Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable** − You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases** − Python provides interfaces to all major commercial databases.
- **GUI Programming** − Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable** − Python provides a better structure and support for large programs than shell scripting.

STANDARD DATA TYPES

The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

Python has five standard data types:

- Numbers
- String
- List
- Tuple
- Dictionary

Python is an open-source programming language that was made to be easy-to-read and powerful. A Dutch programmer named Guido van Rossum made Python in 1991. He named it after the television show Monty Python's Flying Circus. the show.

Python is an interpreted language. Interpreted languages do not need to be compiled to run. A program called an interpreter runs Python code on almost any kind of computer. This means that a programmer can change the code and quickly see the results. This also means Python is slower than a compiled language like C, because it is not running machine code directly.

Python is a good programming language for beginners. It is a high-level language, which means a programmer can focus on what to do instead of how to do it.

Python drew inspiration from other programming languages like C, C++, Java, Perl, and Lisp.

Python has a very easy-to-read syntax. Some of Python's syntax comes from C, because that is the language that Python was written in. But Python uses whitespace to delimit code: spaces or tabs are used to organize code into groups. Its standard library is made up of many functions that come with Python when it is installed. On the Internet there are many other libraries available that make it possible for the Python language to do more things. These libraries make it a powerful language; it can do many different things.

Some things that Python is often used for are:

- Web development
- Scientific programming
- Desktop GUIs
- Network programming
-

PYTHON LIBRARIES ARE

1.Numpy:

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

NumPy's main object is the homogeneous multidimensional array. It is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers. In NumPy dimensions are called axes. The number of axes is rank.

- Offers Matlab-ish capabilities within Python
- Fast array operations
- 2D arrays, multi-D arrays, linear algebra etc.

2.Pandas:

Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data. The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.

3.Matplotlib:

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible. Create publication quality plots. Make interactive figures that can zoom, plan, update.

4.Seaborn:

Python Seaborn library is a widely popular data visualization library that is commonly used for data science and machine learning tasks. You build it on top of the matplotlib data visualization library and can perform exploratory analysis. You can create interactive plots to answer questions about your data

5.Scipy:

SciPy is a scientific computation library that uses NumPy underneath. SciPy stands for Scientific Python. It provides more utility functions for optimization, stats and signal processing. Like NumPy, SciPy is open source so we can use it freely.

Matplotlib: High quality plotting library.

PYTHON MODULES

Python allows us to store our code in files (also called modules). This is very useful for more serious programming, where we do not want to retype a long function definition from the very beginning just to change one mistake. In doing this, we are essentially defining our own modules, just like the modules defined already in the Python library.

To support this, Python has a way to put definitions in a file and use them in a script or in an interactive instance of the interpreter. Such a file is called a module; definitions from a module can be imported into other modules or into the main module.

TESTING CODE

As indicated above, code is usually developed in a file using an editor.

- To test the code, import it into a Python session and try to run it.
- Usually there is an error, so you go back to the file, make a correction, and test again.
- This process is repeated until you are satisfied that the code works.
- The entire process is known as the development cycle.
- There are two types of errors that you will encounter. Syntax errors occur when the form of some command is invalid.
- This happens when you make typing errors such as misspellings, or call something by the wrong name, and for many other reasons. Python will always give an error message for a syntax error.

FUNCTIONS IN PYTHON

It is possible, and very useful, to define our own functions in Python. Generally speaking, if you need to do a calculation only once, then use the interpreter. But when you or others have need to perform a certain type of calculation many times, then define a function**.**

You use functions in programming to bundle a set of instructions that you want to use repeatedly or that, because of their complexity, are better self-contained in a sub-program and called when needed. That means that a function is a piece of code written to carry out a specified task.

To carry out that specific task, the function might or might not need multiple inputs. When the task is carried out, the function can or cannot return one or more values. There are three types of functions in python:

help (), min (), print ().

PYTHON NAMESPACE

A **namespace** (sometimes also called a context) is a naming system for making names unique to avoid ambiguity. Everybody knows a name-spacing system from daily life, i.e. the naming of people in first name and family name (surname).

An example is a network: each network device (workstation, server, printer, ...) needs a unique name and address. Yet another example is the directory structure of file systems.

- The same file name can be used in different directories, the files can be uniquely accessed via the pathnames. Many programming languages use namespaces or contexts for identifiers. An identifier defined in a namespace is associated with that namespace.
- This way, the same identifier can be independently defined in multiple namespaces. (Like the same file names in different directories) Programming languages, which support namespaces, may have different rules that determine to which namespace an identifier belongs.
- Namespaces in Python are implemented as Python dictionaries, this means it is a mapping from names (keys) to objects (values). The user doesn't have to know this to write a Python program and when using namespaces.

Some namespaces in Python:

- **global names** of a module
- **local names** in a function or method invocation
- **built-in names**: this namespace contains built-in functions (e.g. abs(), cmp(), ...) and built-inexception names.

GARBAGE COLLECTION

Garbage Collector exposes the underlying memory management mechanism of Python, the automatic garbage collector. The module includes functions for controlling how the collector operates and to examine the objects known to the system, either pending collection or stuck in reference cycles and unable to be freed.

4.3 GOOGLE COLAB:

Collaboratory (aka Google colab) is a research tool for machine learning education and research. Google Colab is a cloud-based platform provided by Google that allows you to write and execute Python code in a Jupyter Notebook-like interface. It offers free access to GPU and

TPU resources, which can be useful for tasks like machine learning and data analysis that require significant computational power. With Colab, you can write, share, and collaborate on notebooks with others in real-time. The platform also supports Markdown cells for documentation and visualization of data using libraries like Matplotlib and Seaborn. Collaboratory is based on Jupyter and Jupyter notebooks can be used and shared without having to download, install, or run anything on your own computer. Collaboratory supports Python 2.7 and Python 3.6. Data analysis and visualization is frequently done using pandas.

CHAPTER – 5

SYSTEM DESIGN

CHAPTER-5
SYSTEM DESIGN

5.1 INTRODUCTION

Software design sits at the technical kernel of the software engineering process and is applied regardless of the development paradigm and area of application. Design is the first step in the development phase for any engineered product or system. The designer's goal is to produce a model or representation of an entity that will later be built. Beginning, once system requirements have been specified and analyzed, system design is the first of the three technical activities -design, code and test that is required to build and verify software.

The importance can be stated with a single word "Quality". Design is the place where quality is fostered in software development. Design provides us with representations of software that can assess quality. Design is the only way that we can accurately translate a customer's view into a finished software product or system. Software design serves as a foundation for all the software engineering steps that follow. Without a strong design we risk building an unstable system – one that will be difficult to test, one whose quality cannot be assessed until the last stage. The purpose of the design phase is to plan a solution to the problem specified by the requirement document. This phase is the first step in moving from the problem domain to the solution domain. In other words, starting with what is needed, design takes us toward how to satisfy the needs. The design of a system is perhaps the most critical factor affecting the quality of the software; it has a major impact on the later phase, particularly testing, maintenance. The output of this phase is the design document. This document is similar to a blueprint for the solution and is used later during implementation, testing and maintenance. The design activity is often divided into two separate phases System Design and Detailed Design.

System Design also called top-level design aims to identify the modules that should be in the system, the specifications of these modules, and how they interact with each other to produce the desired results. At the end of the system design all the major data structures, file formats, output formats, and the major modules in the system and their specifications are decided.

System Design also called top-level design aims to identify the modules that should be in the system, the specifications of these modules, and how they interact with each other to produce the desired results. At the end of the system design all the major data structures, file formats, output formats, and the major modules in the system and their specifications are decided.

In system design the focus is on identifying the modules, whereas during detailed design the focus is on designing the logic for each of the modules. In other works, in system design the attention is on what components are needed, while in detailed design how the components can be implemented in software is the issue.

Design is concerned with identifying software components specifying relationships among components. Specifying software structure and providing blue print for the document phase. Modularity is one of the desirable properties of large systems. It implies that the system is divided into several parts. In such a manner, the interaction between parts is minimal clearly specified.

During the system design activities, Developers bridge the gap between the requirements specification, produced during requirements elicitation and analysis, and the system that is delivered to the user.

Design is the place where quality is fostered in development. Software design is a process through which requirements are translated into a representation of software.

5.2 UML Diagram Overview

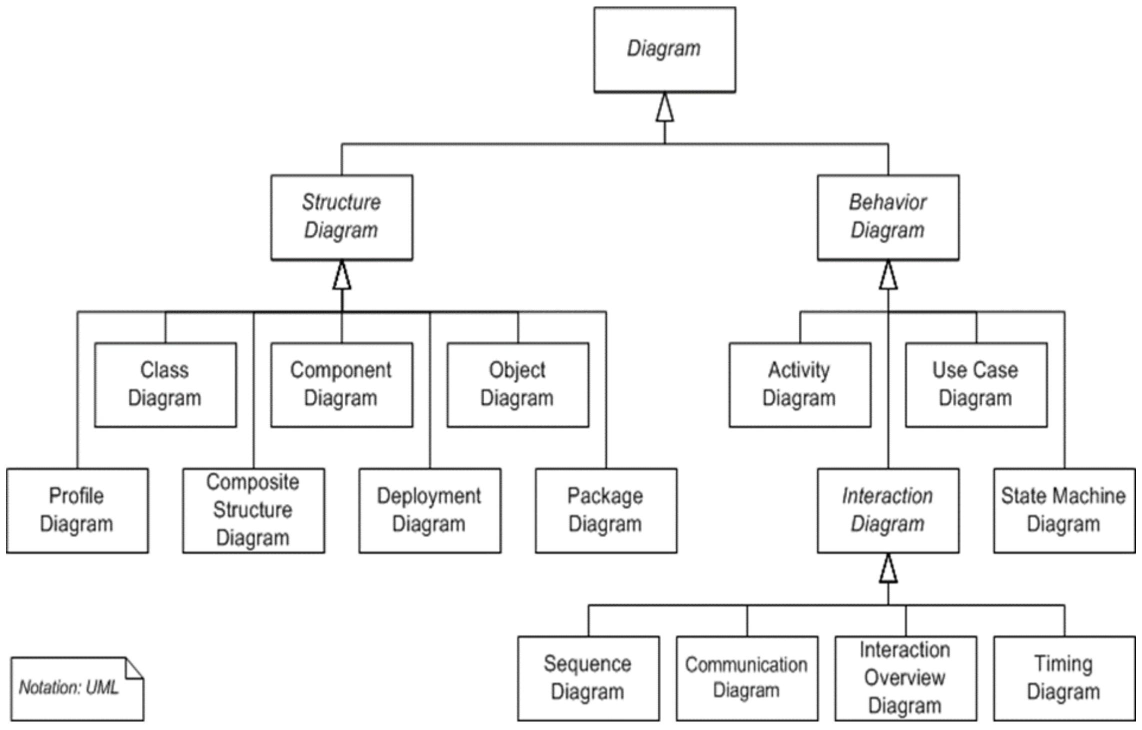


Fig.4: UML Diagram Overview

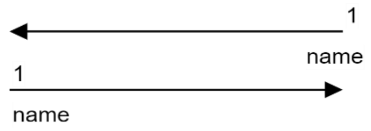
UML combines the best techniques from data modeling (entity relationship diagrams), business modeling (workflows), object modeling, and component modeling. It can be used with all processes, throughout the software development life cycle, and across different implementation technologies. UML has synthesized the notations of the Booch method, the Object-modeling technique (OMT) and Object-oriented software engineering (OOSE) by fusing them into a single, common and widely usable modeling language. UML aims to be a standard modeling language which can model concurrent and distributed systems.

5.3 DATA FLOW DIAGRAMS

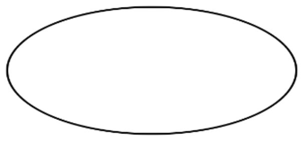
A graphical tool used to describe and analyze the moment of data through a system manual or automated including the process, stores of data, and delays in the system. Data Flow Diagrams are the central tool and the basis from which other components are developed. The transformation of data from input to output, through processes, may be described logically and independently of the physical components associated with the system. The DFD is also know as a data flow graph or a bubble chart.

DFDs are the model of the proposed system. They clearly should show the requirements on which the new system should be built. Later during design activity this is taken as the basis for drawing the system's structure charts. The Basic Notation used to create a DFD's are as follows:

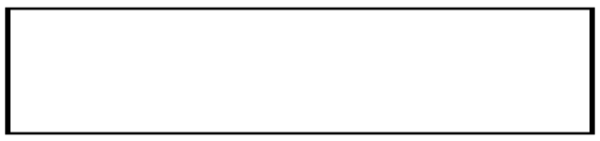
Dataflow: Data moves in a specific direction from an origin to a destination.



Process: People, procedures, or devices that use or produce (Transform) Data. The physical component is not identified.



Source: External sources or destination of data, which may be People, programs, organizations or other entities.



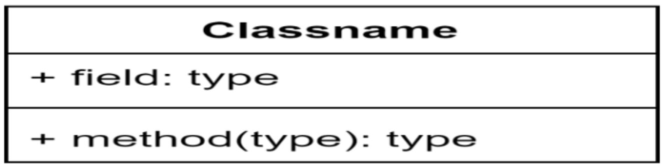
What is a UML Class Diagram?

Class diagrams are the backbone of almost every object-oriented method including UML. They describe the static structure of a system.

Basic Class Diagram Symbols and Notations

Classes represent an abstraction of entities with common characteristics. Associations represent the relationships between classes.

Illustrate classes with rectangles divided into compartments. Place the name of the class in the first partition (centered, bolded, and capitalized), list the attributes in the second partition, and write operations into the third.



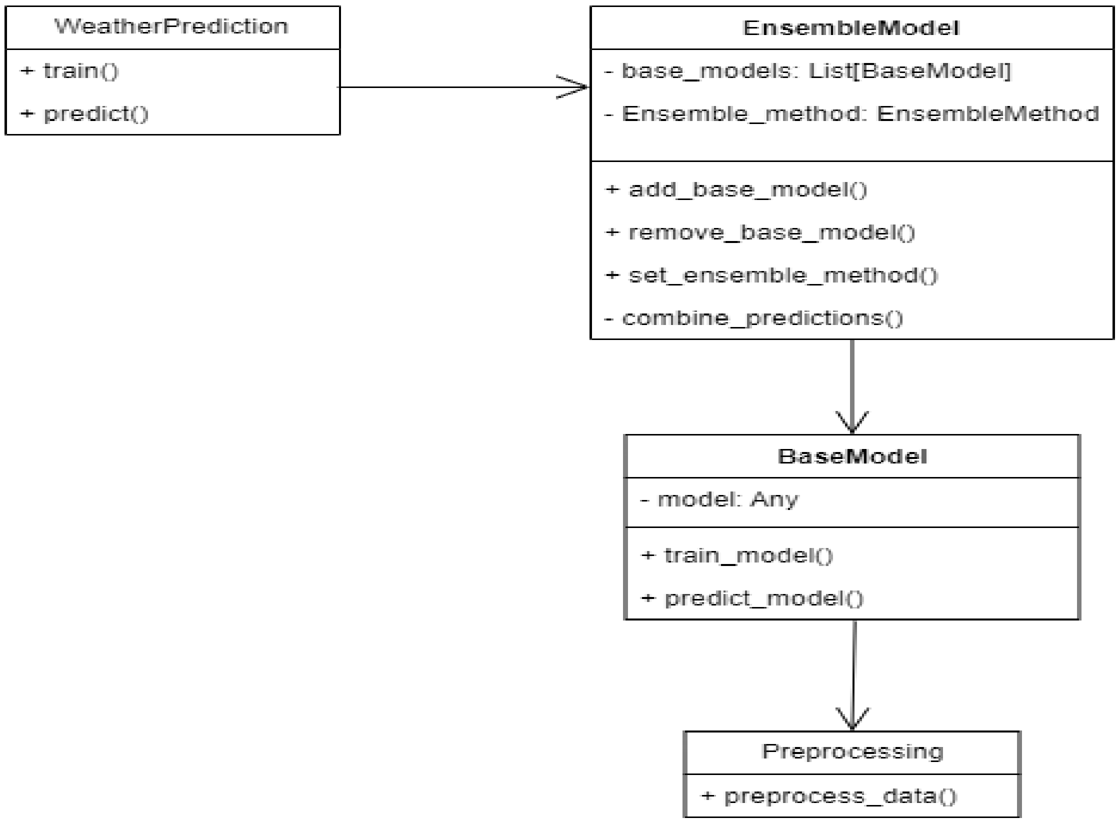
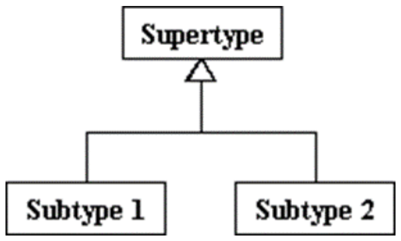


Fig.5.1: Class Diagram

Generalization

Generalization is another name for inheritance or an "is a" relationship. It refers to a relationship between two classes where one class is a specialized version of another. For example, Honda is a type of car. So the class Honda would have a generalization relationship with the class car.



In real life coding examples, the difference between inheritance and aggregation can be confusing. If you have an aggregation relationship, the aggregate (the whole) can access only the PUBLIC functions of the part class. On the other hand, inheritance allows the inheriting class to access both the PUBLIC and PROTECTED functions of the super class.

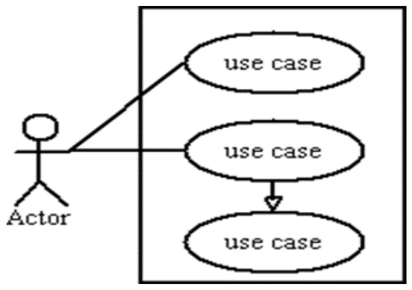
What is a UML Use Case Diagram?

Use case diagrams model the functionality of a system using actors and use cases. Use cases are services or functions provided by the system to its users.

Basic Use Case Diagram Symbols and Notations

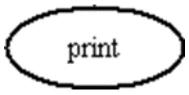
System

Draw your system's boundaries using a rectangle that contains use cases. Place actors outside the system’s boundaries.



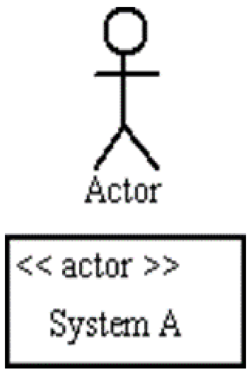
Use Case

Draw use cases using ovals. Label with ovals with verbs that represent the system's functions.



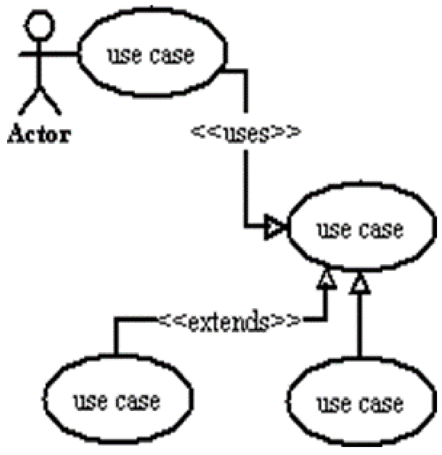
Actors

Actors are the users of a system. When one system is the actor of another system, label the actor system with the actor stereotype.



Relationships

Illustrate relationships between an actor and a use case with a simple line. For relationships among use cases, use arrows labeled either "uses" or "extends." A "uses" relationship indicates that one use case is needed byanother in order to perform a task. An "extends" relationship indicates alternative options under a certain usecase.



Sequence Diagram

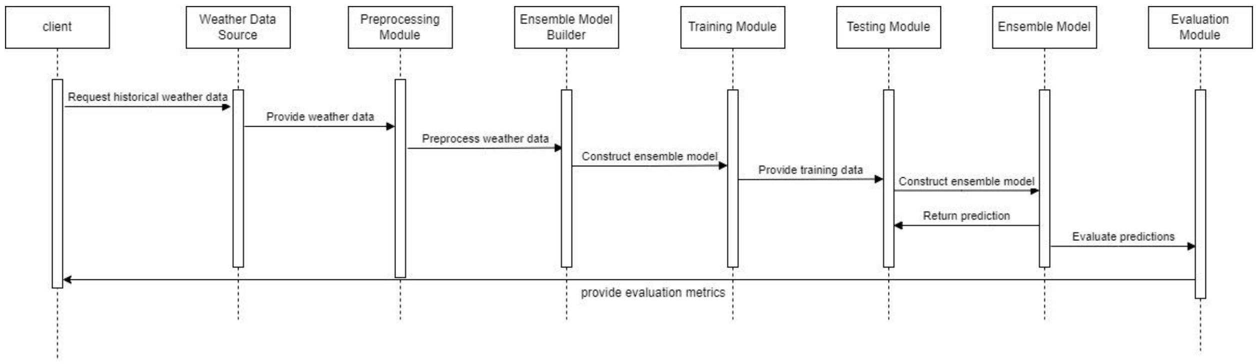


Fig.5.2: Sequence Diagram

Creating a sequence diagram for weather prediction using ensemble methods involves illustrating the flow of interactions between different components or objects involved in the process. Here's a simplified sequence diagram for such a scenario:

Client:

- Initiates the weather prediction process.

Weather Data Source:

- Provides historical weather data to the system.

Preprocessing Module:

- Receives weather data from the data source.
- Performs data preprocessing tasks such as cleaning, transforming, and feature engineering.

Ensemble Model Builder:

- Constructs the ensemble model.
- Select individual base models (e.g., Random Forest, Gradient Boosting, etc.) for the ensemble.
- Combines the base models using a specific ensemble method (e.g., Voting, Stacking, etc.).

Training Module:

- Receives preprocessed weather data and the ensemble model.
- Splits the data into training and validation sets.
- Trains each base model on the training data.

- Integrates individual base models into the ensemble.

Testing Module:

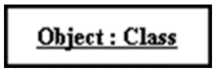
- Receives unseen test data.
- Applies the trained ensemble model to make predictions.
- Outputs the predicted weather parameters (e.g., temperature).

Evaluation Module:

- Calculates evaluation metrics such as R-squared, RMSE, etc.
- Compares predicted weather values with actual observations.
- Provides feedback on the performance of the ensemble model.**Basic Sequence Diagram Symbols and Notations**

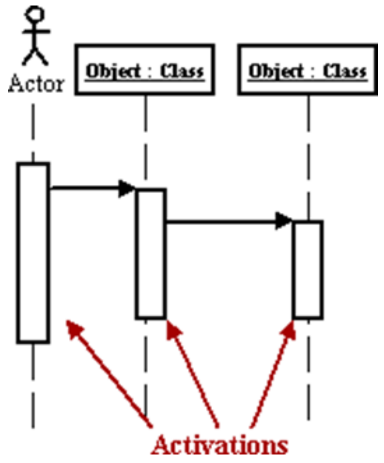
Class role

Class roles describe the way an object will behave in context. Use the UML object symbol to illustrate class roles but, don't list object attributes.



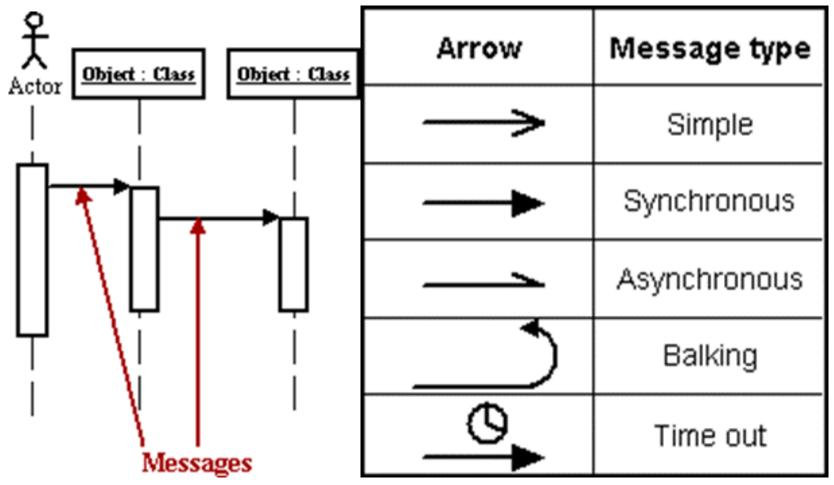
Activation

Activation boxes represent the time an object needs to complete a task.



Messages

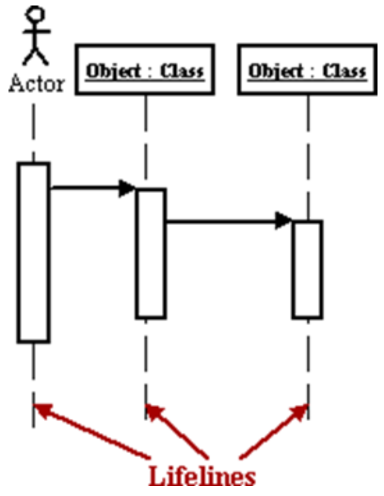
Messages are arrows that represent communication between objects. Use half-arrowed lines to represent asynchronous messages. Asynchronous messages are sent from an object that will not wait for a response from the receiver before continuing its tasks.



Various message types for Sequence and Collaboration diagrams

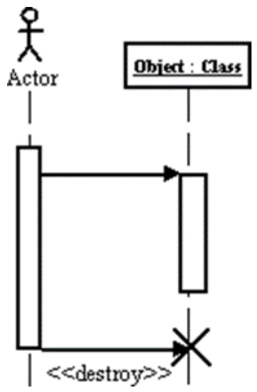
Lifelines

Various message types for Sequence and Collaboration diagrams



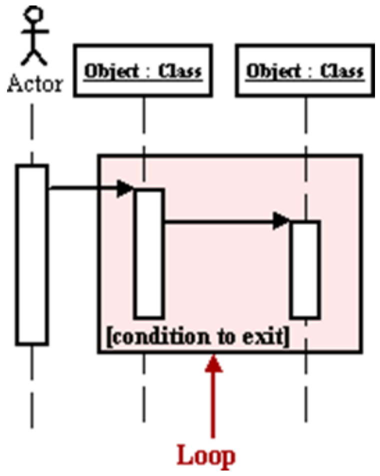
Destroying Objects

Objects can be terminated early using an arrow labeled "<< destroy>>" that points to an X.



Loops

A repetition or loop within a sequence diagram is depicted as a rectangle. Place the condition for exiting the loop at the bottom left corner in square brackets [ ].



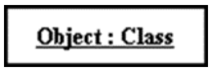
Collaboration Diagram

A collaboration diagram describes interactions among objects in terms of sequenced messages. Collaboration diagrams represent a combination of information taken from class, sequence, and use case diagrams describing both the static structure and dynamic behavior of a system.

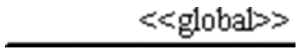
Basic Collaboration Diagram Symbols and Notations

Class roles

Class roles describe how objects behave. Use the UML object symbol to illustrate class roles, but don't list object attributes.

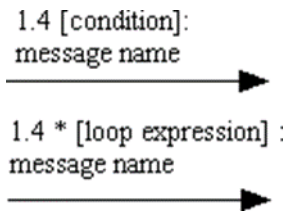


Association roles
Association roles describe how an association will behave given a particular situation. You can draw association roles using simple lines labeled with stereotypes.



Messages

Unlike sequence diagrams, collaboration diagrams do not have an explicit way to denote time and instead number messages in order of execution. Sequence numbering can become nested using the Dewey decimal system. For example, nested messages under the first message are labeled 1.1, 1.2, 1.3, and so on. The a condition for a message is usually placed in square brackets immediately following the sequence number. Use a * after the sequence number to indicate a loop.



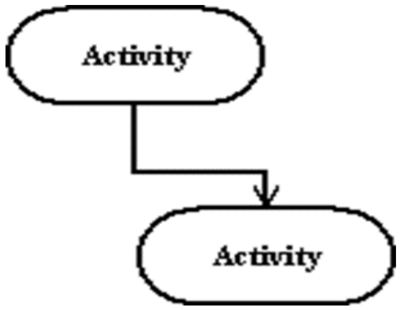
Activity Diagram

An activity diagram illustrates the dynamic nature of a system by modeling the flow of control from activity to activity. An activity represents an operation on some class in the system that results in a change in the state of the system. Typically, activity diagrams are used to model workflow or business processes and internal operation. Because an activity diagram is a special kind of state chart diagram, it uses some of the same modeling conventions.

<u>Basic Activity Diagram Symbols and Notations</u>

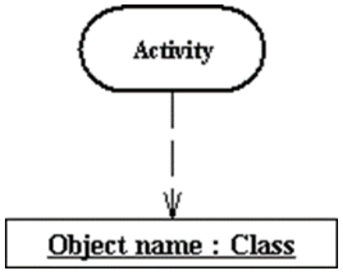
Action states
Action states represent the non-interruptible actions of objects. You can draw an action state in Smart Draw using a rectangle with rounded corners.

Action Flow:
Action flow arrows illustrate the relationships among action states.



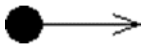
Object Flow

Object flow refers to the creation and modification of objects by activities. An object flow arrow from an action to an object means that the action creates or influences the object. An object flow arrow from an object to an action indicates that the action state uses the object.

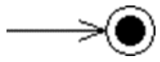


Initial State

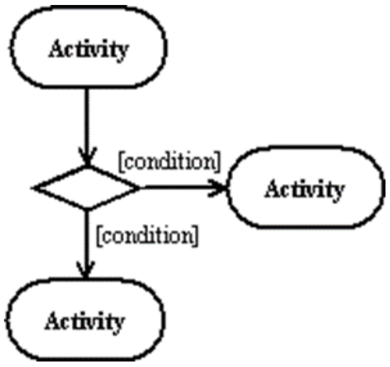
A filled circle followed by an arrow represents the initial action state.



Final State
An arrow pointing to a filled circle nested inside another circle represents the final action state.

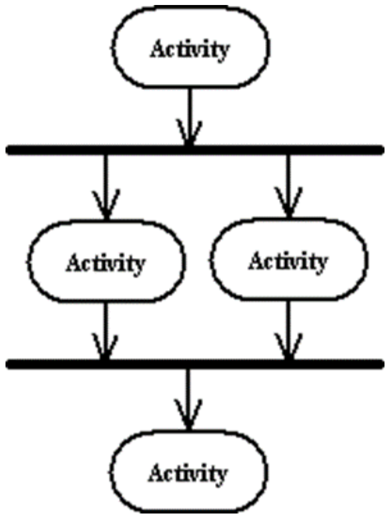


Branching
A diamond represents a decision with alternate paths. The outgoing alternates should be labeled with a condition or guard expression. You can also label one of the paths "else."



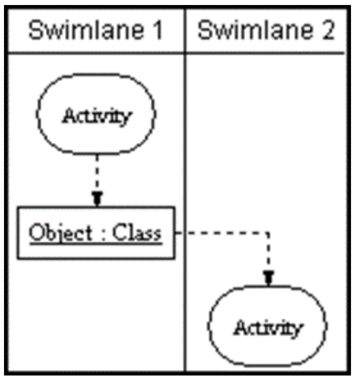
Synchronization

A synchronization bar helps illustrate parallel transitions. Synchronization is also called forking and joining.



Swimlanes

Swimlanes group related activities into one column.



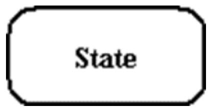
State chart Diagram

A state chart diagram shows the behavior of classes in response to external stimuli. This diagram models the dynamic flow of control from state to state within a system.

Basic State chart Diagram Symbols and Notations

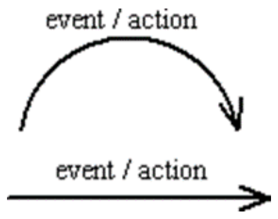
States

States represent situations during the life of an object. You can easily illustrate a state in Smart Draw by using a rectangle with rounded corners.



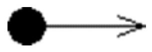
Transition

A solid arrow represents the path between different states of an object. Label the transition with the event that triggered it and the action that results from it.



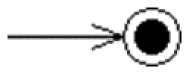
Initial State

A filled circle followed by an arrow represents the object's initial state.



Final State

An arrow pointing to a filled circle nested inside another circle represents the object's final state.



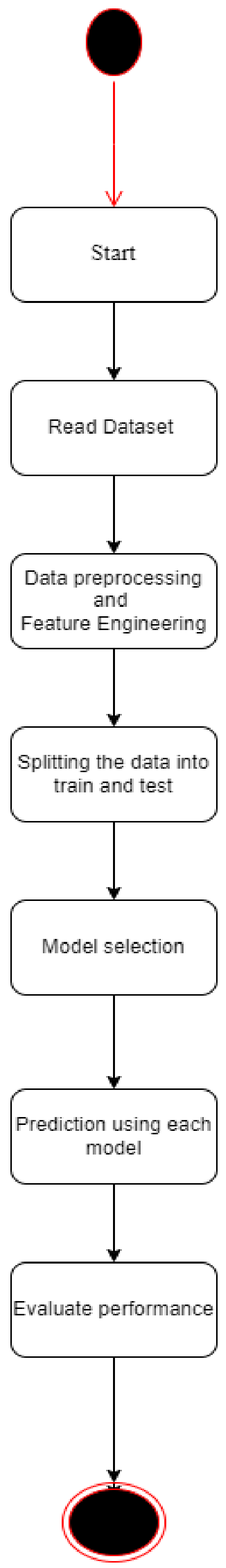

Start
Read Dataset
Data preprocessing and Feature Engineering
Splitting the data into train and test
Model selection
Prediction using each model
Evaluate performance

Fig.5.3: State Diagram

This diagram represents the workflow of the script:

- Starting with reading the dataset.
- Preprocessing the data, including handling missing values, converting date formats, dropping unnecessary columns, and label encoding.
- Feature engineering to extract year, month, and day from the date.
- Splitting the data into training and testing sets.
- Selecting different regression models for training: GradientBoosting, RandomForest, KNeighbors, DecisionTree, SVR, and a VotingRegressor ensemble.
- Predicting using each trained model and evaluating their performance.
- Finally, implementing Stacking Regression, predicting with the stacked model, and evaluating its performance.

MODEL DESIGN

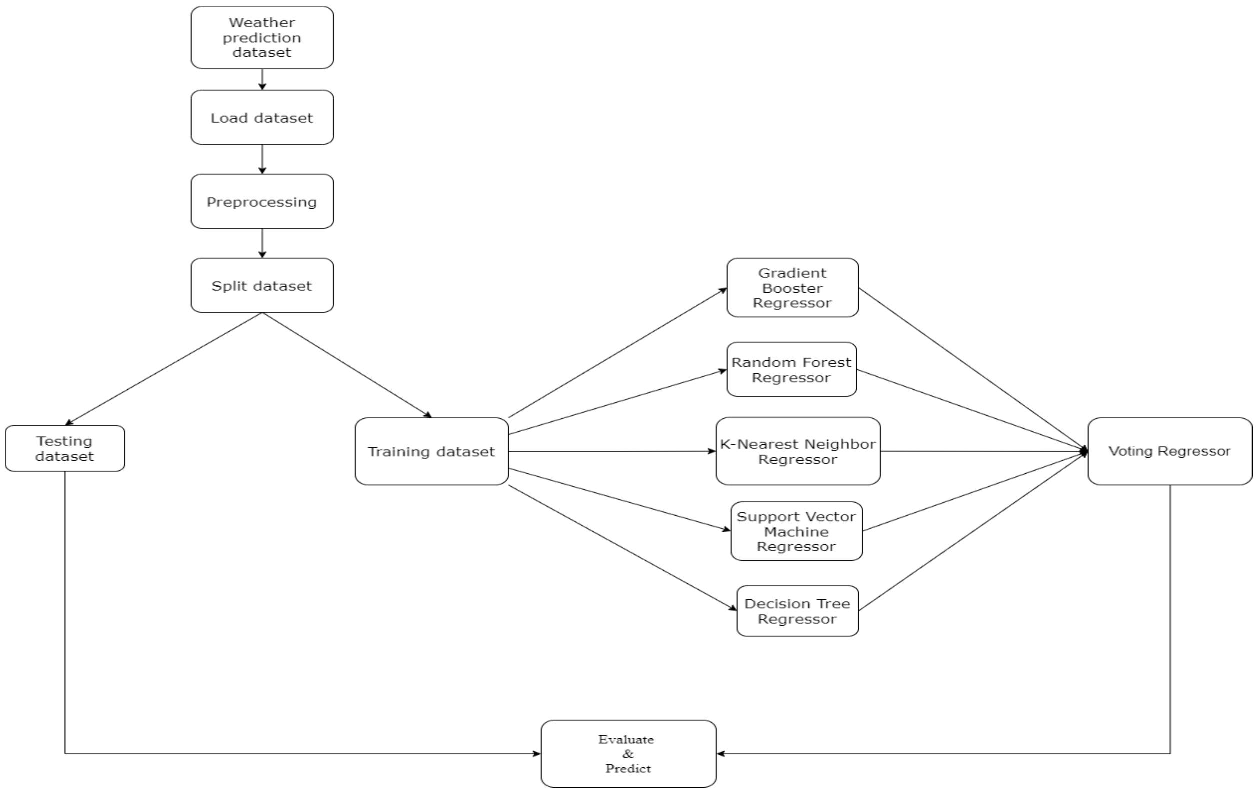


Fig.6.Flow diagram

CHAPTER-6
IMPLEMENTATION

CHAPTER-6

IMPLEMENTATION

```
import numpy as np
import pandas as pd
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import VotingRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn.preprocessing import *
dataset = pd.read_csv("/content/weatherHistory.csv")
dataset.head()
dataset.isnull().sum() #there are 517 null values which should be replaced
dataset["Formatted Date"] = pd.to_datetime(dataset["Formatted Date"],
format = "%Y-%m-%d %H:%M:%S.%f %z")
#"loud cover" has only one unique value
dataset = dataset.drop(["Loud Cover","Daily Summary"], axis=1)
dataset.corr()
#apparent temperature is highly correlated to temperature and should be
removed
```

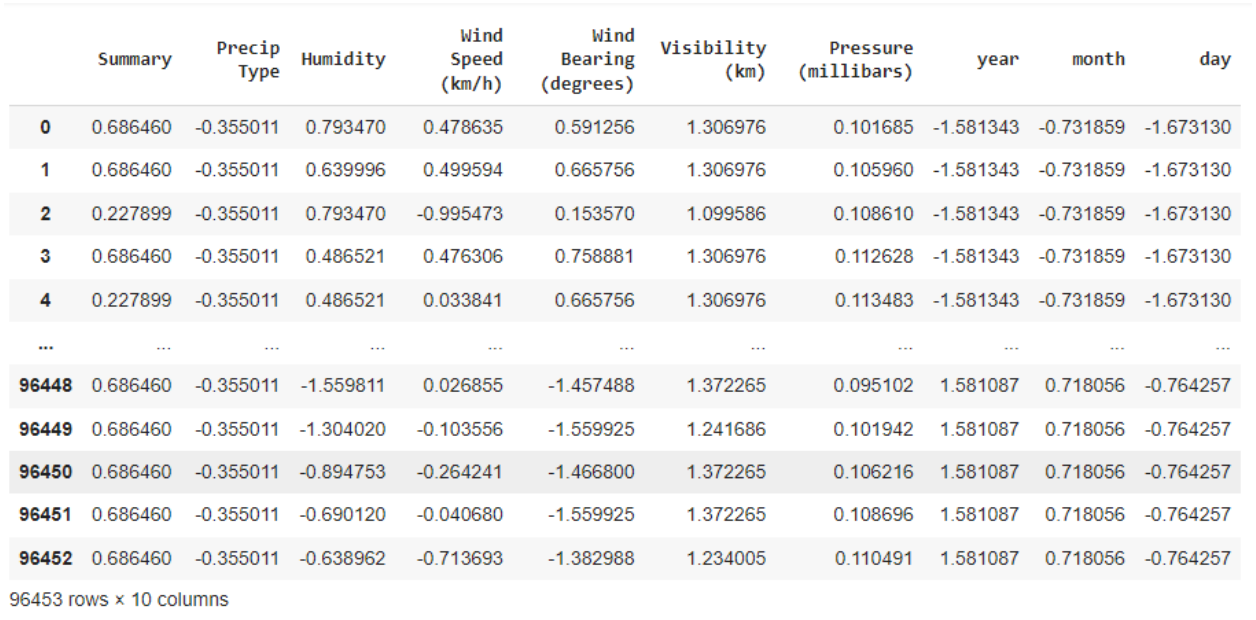
```
dataset = dataset.drop(["Apparent Temperature (C)"], axis=1)
X = dataset
dataset["year"] = dataset["Formatted Date"].apply(lambda x: x.year)
dataset["month"] = dataset["Formatted Date"].apply(lambda x: x.month)
dataset["day"] = dataset["Formatted Date"].apply(lambda x: x.day)
dataset = dataset.drop(["Formatted Date"], axis=1)
#Label Encoding Text Values
le = LabelEncoder()
le.fit(dataset["Summary"])
dataset["Summary"] = le.transform(dataset["Summary"])
le.fit(dataset["Precip Type"])
dataset["Precip Type"] = le.transform(dataset["Precip Type"])
y = dataset["Temperature (C)"]
X = dataset.drop(["Temperature (C)"], axis = 1)
X
#test and train data set split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,
random_state=40)
print(X_train)
print(y_train)
gbr=GradientBoostingRegressor()
rfr=RandomForestRegressor()
knr=KNeighborsRegressor()
dtr=DecisionTreeRegressor()
svr=SVR()
```

```
vr=VotingRegressor([('gbr',gbr),('rfr',rfr),('knr',knr),('dtr',dtr),('svr',svr)])
print(np.isnan(X_train).sum())
print(np.isnan(y_train).sum())
X_train[np.isnan(X_train)] = 0  # Replace NaN values with 0 (you can choose
a different strategy if needed)
y_train[np.isnan(y_train)] = 0
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
vr.fit(X_train,y_train)
y_pred = vr.predict(X_test)
from sklearn.metrics import r2_score
r2_score(y_test, y_pred)
from sklearn.metrics import mean_squared_error
np.sqrt(mean_squared_error(y_test,y_pred))
gbr.fit(X_train,y_train)
rfr.fit(X_train,y_train)
knr.fit(X_train,y_train)
dtr.fit(X_train,y_train)
svr.fit(X_train,y_train)
from sklearn.model_selection import cross_val_score
cv_scores = cross_val_score(rfr, X_train, y_train, cv=5, scoring='r2')
print("Cross-Validation R-squared scores:", cv_scores)
print("Mean CV R-squared:", np.mean(cv_scores))
v1=r2_score(y_test, gbr.predict(X_test))
v2=r2_score(y_test, rfr.predict(X_test))
v3=r2_score(y_test, knr.predict(X_test))
```

```
v4=r2_score(y_test, dtr.predict(X_test))
v5=r2_score(y_test, svr.predict(X_test))
v6=r2_score(y_test, y_pred)
print(v1)
print(v2)
print(v3)
print(v4)
print(v5)
print(v6)
model_names = ['GradientBoosting', 'RandomForest', 'KNeighbors',
'DecisionTree', 'SVR', 'Voting']
r2_values = [v1, v2, v3, v4, v5, v6]
rmse_values = [sqrt(mean_squared_error(y_test, model.predict(X_test))) for
model in [gbr, rfr, knr, dtr, svr, vr]]
import matplotlib.pyplot as plt
fig, ax = plt.subplots(1, 2, figsize=(14, 6))
ax[0].bar(model_names, r2_values)
ax[0].set_title('R-squared Comparison')
ax[0].set_ylabel('R-squared')
ax[1].bar(model_names, rmse_values)
ax[1].set_title('RMSE Comparison')
ax[1].set_ylabel('RMSE')
plt.tight_layout()
plt.show()
from sklearn.ensemble import StackingRegressor
from sklearn.metrics import r2_score, mean_squared_error
from math import sqrt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
# Define base models
base_models = [
    ('gbr', GradientBoostingRegressor()),
    ('rfr', RandomForestRegressor()),
    ('knr', KNeighborsRegressor()),
    ('dtr', DecisionTreeRegressor()),
    ('svr', SVR())
]
# Initialize stacking regressor
stacking_reg = StackingRegressor(estimators=base_models,
final_estimator=LinearRegression())
# Train stacking regressor
stacking_reg.fit(X_train, y_train)
# Make predictions
y_pred_stacking = stacking_reg.predict(X_test)
print(y_pred_stacking)
# Evaluate the performance
r2_stacking = r2_score(y_test, y_pred_stacking)
rmse_stacking = np.sqrt(mean_squared_error(y_test, y_pred_stacking))
print("Stacking R-squared:", r2_stacking)
print("Stacking RMSE:", rmse_stacking)
import matplotlib.pyplot as plt
```

```
# R-squared values
r2_values = [v1, v2, v3, v4, v5, v6, r2_stacking]
model_names.append('Stacking')  # Add Stacking Regressor to model names

# RMSE values
rmse_values.append(rmse_stacking)

# Plotting
fig, ax = plt.subplots(1, 2, figsize=(14, 6))

# R-squared comparison
ax[0].bar(model_names, r2_values)
ax[0].set_title('R-squared Comparison')
ax[0].set_ylabel('R-squared')
ax[0].axhline(y=0, color='black', linewidth=1)  # Add a horizontal line at y=0
for reference

# RMSE comparison
ax[1].bar(model_names, rmse_values)
ax[1].set_title('RMSE Comparison')
ax[1].set_ylabel('RMSE')

plt.tight_layout()
plt.show()
```

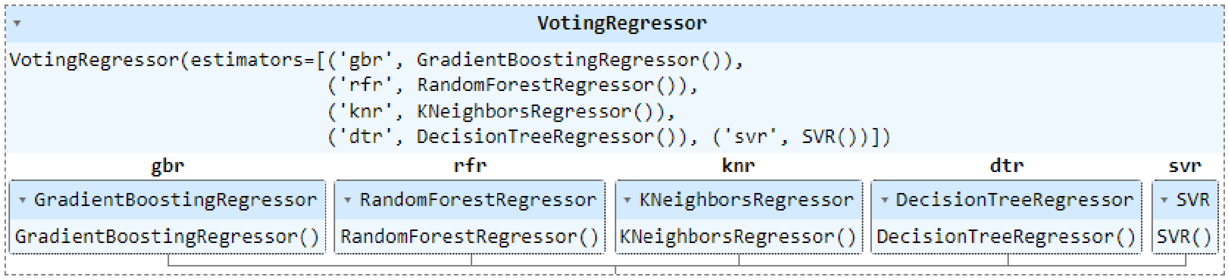
RESULTS



	Summary	Precip Type	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Pressure (millibars)	year	month	day
0	0.686460	-0.355011	0.793470	0.478635	0.591256	1.306976	0.101685	-1.581343	-0.731859	-1.673130
1	0.686460	-0.355011	0.639996	0.499594	0.665756	1.306976	0.105960	-1.581343	-0.731859	-1.673130
2	0.227899	-0.355011	0.793470	-0.995473	0.153570	1.099586	0.108610	-1.581343	-0.731859	-1.673130
3	0.686460	-0.355011	0.486521	0.476306	0.758881	1.306976	0.112628	-1.581343	-0.731859	-1.673130
4	0.227899	-0.355011	0.486521	0.033841	0.665756	1.306976	0.113483	-1.581343	-0.731859	-1.673130
...	...	...	...	...	...	...	...	...	...	...
96448	0.686460	-0.355011	-1.559811	0.026855	-1.457488	1.372265	0.095102	1.581087	0.718056	-0.764257
96449	0.686460	-0.355011	-1.304020	-0.103556	-1.559925	1.241686	0.101942	1.581087	0.718056	-0.764257
96450	0.686460	-0.355011	-0.894753	-0.264241	-1.466800	1.372265	0.106216	1.581087	0.718056	-0.764257
96451	0.686460	-0.355011	-0.690120	-0.040680	-1.559925	1.372265	0.108696	1.581087	0.718056	-0.764257
96452	0.686460	-0.355011	-0.638962	-0.713693	-1.382988	1.234005	0.110491	1.581087	0.718056	-0.764257

96453 rows × 10 columns

X_train shape: (67517, 10)
y_train shape: (67517,)



```
VotingRegressor
VotingRegressor(estimators=[('gbr', GradientBoostingRegressor()),
                            ('rfr', RandomForestRegressor()),
                            ('knr', KNeighborsRegressor()),
                            ('dtr', DecisionTreeRegressor()), ('svr', SVR())])
gbr                          rfr                      knr                    dtr                      svr
GradientBoostingRegressor    RandomForestRegressor    KNeighborsRegressor    DecisionTreeRegressor    SVR
GradientBoostingRegressor()  RandomForestRegressor()  KNeighborsRegressor()  DecisionTreeRegressor()  SVR()
```

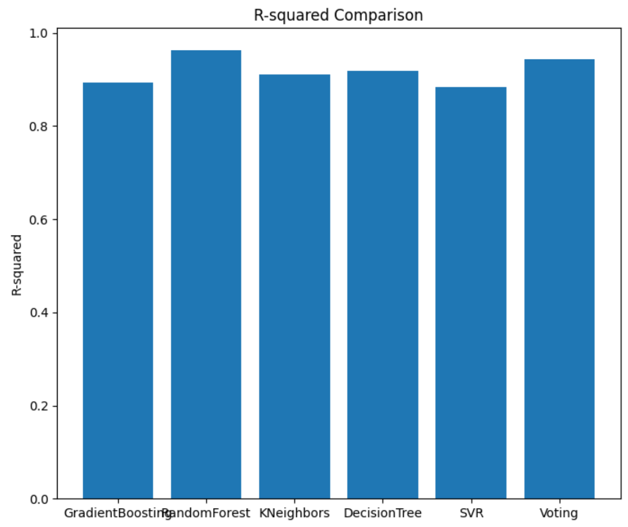
# R-squared (r2_score)
0.9437084335135322

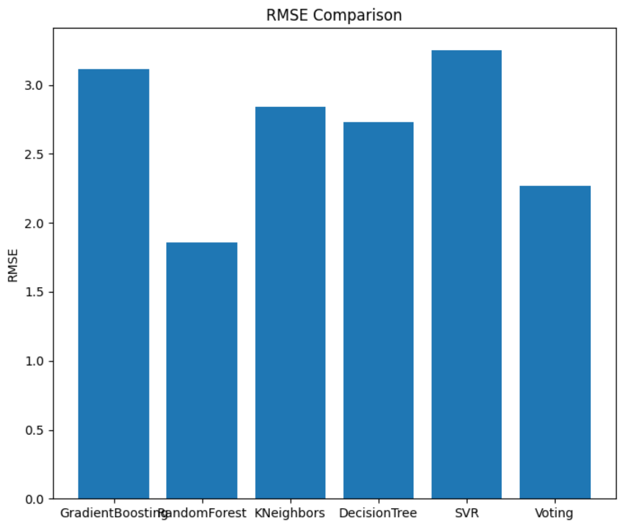
# Root Mean Squared Error (RMSE).
2.2710382619422353

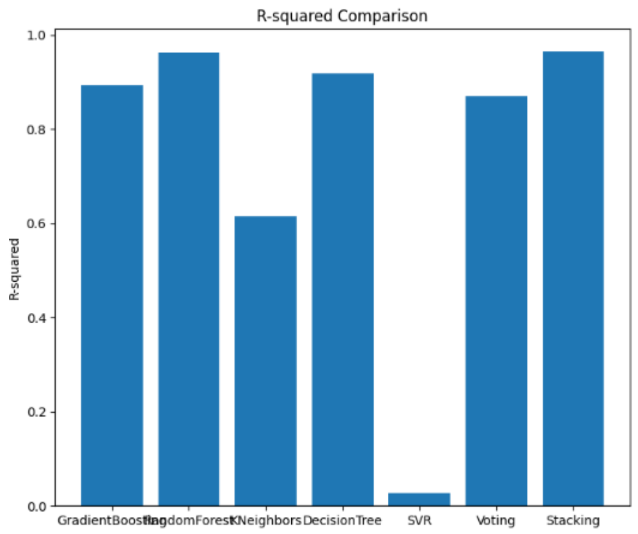
Cross-Validation R-squared scores: [0.95865403 0.95684863 0.95909058 0.95918811 0.959044]

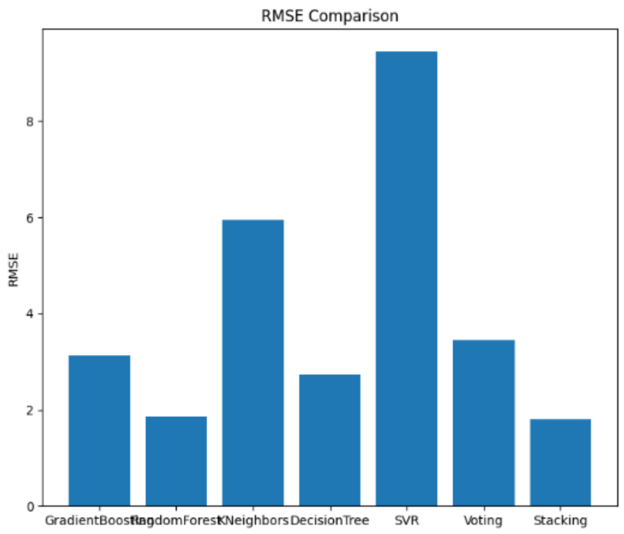
Mean CV R-squared: 0.9585650697641979

```
# r2_score
0.8938905921424415
0.9622602676538342
0.91169703926263
0.9187327902969951
0.8847043071787937
0.9437084335135322
```









CHAPTER - 7

SYSTEM TESTING

CHAPTER-7
SYSTEM TESTING

Introduction to testing:

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies, and/or a finished product. It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail unacceptably. There are various types of tests. Each test type addresses a specific testing requirement.

Types of Testing:

- **Unit Testing**
  Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application.it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at the component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.
- **Integration Testing**
  Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfied, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.
- **Functional Test**

  Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

- Valid Input: identified classes of valid input must be accepted.
- Invalid Input: identified classes of invalid input must be rejected.
- Functions: identified functions must be exercised.
- Output: identified classes of application outputs must be exercised.
- Systems/Procedures: interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage aboutidentifying Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined

- **System Test**
  - System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration-oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

- **White Box Testing**
  - White Box Testing is a testing in which the software tester knows the inner workings, structure and language of the software, or at least its purpose. It is purposeful. It is used to test areas that cannot be reached from a black box level.

- **Black Box Testing**
  - Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated as a black box. You cannot "see" into it. The test provides inputs and responds to outputs without considering how the software works.

Test cases and test reports:

Unit testing

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.
Test strategy and approach

Field testing will be performed manually and functional tests will be written in detail.

Integration testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software systemor – one step up – software applications at the company level – interact without error.

Test Results:

All thetest cases mentioned above passed successfully. No defects encountered.

Acceptance testing:

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirement.

Test Results:

All the test cases mentioned above passed successfully. No defects encountered

CHAPTER – 8
CONCLUSION

CHAPTER-8

CONCLUSION

8.1 CONCLUSION

Machine learning methods will be a key feature in future weather forecasting and climate analysis, according to the authors' analysis of the literature .The authors identified the most common topics of interest in the field, including photovoltaic and wind energy, atmospheric physics and processes, parametrizations, extreme events, and climate change .The most commonly examined meteorological fields were wind, precipitation, temperature, pressure, and radiation .Artificial Neural Networks (ANN) and Deep Learning (DL) were the most commonly used machine learning methods, followed by Random Forest (RF), GBoost (GB), and Support Vector Regresion (SVR) .The authors also highlighted the potential of machine learning methods in improving weather forecasting accuracy, ensemble forecasting interpretation, data assimilation, and parametrization emulation. These technologies aid in pattern recognition, data analysis, and the development of probabilistic forecasting models, offering promising avenues for further refinement. In conclusion, weather forecasting continues to evolve, with advancements in technology and science continuously enhancing our predictive capabilities.

The code provides a comprehensive analysis of various regression models and ensemble techniques for temperature prediction. It demonstrates how to preprocess the data, train multiple models, evaluate their performance, and compare them using appropriate metrics. Both individual models and ensemble methods like Voting and Stacking are employed to improve predictive accuracy. The final evaluation of the Stacking Regressor provides insights into its performance compared to individual models and the Voting Regressor.

Base models are defined (the same ones used in the Voting Regressor). A Stacking Regressor is initialized with the base models and a final estimator (Linear Regression). The Stacking Regressor is trained on the training data. Predictions are made on the test data using the Stacking Regressor. Performance metrics (R-squared and RMSE) are computed for the Stacking Regressor.

CHAPTER – 9

FUTURE ENHANCEMENT

CHAPTER-9

FUTURE ENHANCEMENT

Instead of using a simple LinearRegression as the final estimator, more complex models or ensemble methods could be explored.Feature selection techniques could be applied to improve the performance of the StackingRegressor.

Hyperparameter tuning could be applied to optimize the performance of individual models.Other ensemble techniques like Bagging or AdaBoost could be explored.One-hot encoding could be considered instead of label encoding, especially if there are more than two categories in the categorical variables.

Additional feature engineering techniques such as creating interaction terms or polynomial features could be explored to capture complex relationships between feature

Model Interpretability: Implement techniques to interpret model predictions, such as feature importance analysis for ensemble models or partial dependence plots to understand the relationship between features and the target variable.

Deployment: Once the model is trained and optimized, consider deploying it into a production environment, such as a web application or an API, to make predictions on new data.

Pipeline Optimization: Organize your code into reusable functions or pipelines to streamline the preprocessing, modeling, and evaluation process. This can improve code readability, maintainability, and scalability.

Error Analysis: Conduct an in-depth analysis of prediction errors to identify patterns or systematic biases in the models and potentially address them through further data preprocessing or model refinement. By incorporating these enhancements, you can further improve the performance, interpretability, and usability of your regression models for weather prediction.

CHAPTER – 10

BIBILOGRAPHY

CHAPTER-10

BIBLIOGRAPHY

1. Singh, Nitin, Saurabh Chaturvedi, and Shamim Akhter. "Weather forecasting using a machine learning algorithm." 2019 International Conference on Signal Processing and Communication (ICSC). IEEE, 2019.

2. Bochenek, Bogdan, and Zbigniew Ustrnul. "Machine learning in weather prediction and climate analyses—applications and perspectives." Atmosphere 13.2 (2022): 180.

3. Ren, Xiaoli, et al. "Deep learning-based weather prediction: a survey." Big Data Research 23 (2021): 100178.

4. Scher, Sebastian, and Gabriele Messori. "Ensemble methods for neural network-based weather forecasts." Journal of Advances in Modeling Earth Systems 13.2 (2021).

5. Wu, Hao, and David Levinson. "The ensemble approach to forecasting: A review and synthesis." Transportation Research Part C: Emerging Technologies 132 (2021): 103357.

6. Gneiting, Tilmann, and Adrian E. Raftery. "Weather forecasting with ensemble methods." Science 310.5746 (2005): 248-249.

7. Ahmadi, Abbas, et al. "Hybrid model for weather forecasting using ensemble of neural networks and mutual information." 2014 IEEE geoscience and remote sensing symposium. IEEE, 2014.

8. Delle Monache, Luca, et al. "Probabilistic weather prediction with an analog ensemble." Monthly Weather Review 141.10 (2013): 3498-3516.

9. Phillips, N. A. "Models for weather prediction." Annual Review of Fluid Mechanics 2.1 (1970): 251-292.

10. Cho, Dongjin, et al. "Comparative assessment of various machine learning-based bias correction methods for numerical weather prediction model forecasts of extreme air temperatures in urban areas." Earth and Space Science 7.4 (2020): e2019EA000740.